AngularJS Introduction

AngularJS is a **JavaScript framework**. It can be added to an HTML page with a <script> tag.

AngularJS extends HTML attributes with **Directives**, and binds data to HTML with **Expressions**.

AngularJS Extends HTML

AngularJS extends HTML with **ng-directives**.

The **ng-app** directive defines an AngularJS application.

The **ng-model** directive binds the value of HTML controls (input, select, textarea) to application data.

The **ng-bind** directive binds application data to the HTML view.

AngularJS Example

```
<!DOCTYPE html>
<html>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min
.js"></script>
<body>

<div ng-app="">
  <p>Name: <input type="text" ng-model="name"></p>
  <p ng-bind="name"></p>
</div>

</body>
</html>
```

Try it Yourself »

Example explained:

AngularJS starts automatically when the web page has loaded.

The **ng-app** directive tells AngularJS that the <div> element is the "owner" of an AngularJS **application**.

The **ng-model** directive binds the value of the input field to the application variable **name**.

The **ng-bind** directive binds the **innerHTML** of the <p> element to the application variable **name**.

AngularJS Directives

As you have already seen, AngularJS directives are HTML attributes with an **ng** prefix. The **ng-init** directive initializes AngularJS application variables.

AngularJS Example

```
<!DOCTYPE html> <html>

<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>

<body>

<div ng-app="" ng-init="firstName='John'">

<p>The name is <span ng-bind="firstName"></span></p>

</div> </body> </html>
```

Alternatively with valid HTML:

AngularJS Example

```
<!DOCTYPE html>

<html>

<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>

<body>

<div data-ng-app="" data-ng-init="firstName='John'">

<p>The name is <span data-ng-bind="firstName"></span></p> </div> </body> </html>
```

AngularJS Expressions

AngularJS expressions are written inside double braces: **{{ expression }}**.

AngularJS will "output" data exactly where the expression is written:

AngularJS Example

```
<!DOCTYPE html>
<html>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min
.js"></script>
<body>

<div ng-app="">
  <p>My first expression: {{ 5 + 5 }}</p>
</div>

</body>
</html>
```

Try it Yourself »

AngularJS expressions bind AngularJS data to HTML the same way as the **ng-bind** directive.

AngularJS Example

```
<!DOCTYPE html>
<html>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min
.js"></script>
<body>

<div ng-app="">
  <p>Name: <input type="text" ng-model="name"></p>
  <p>{{name}}</p>
</div>

</body>
</html>
```

Try it Yourself »

You will learn more about expressions later in this tutorial.

AngularJS Applications

AngularJS **modules** define AngularJS applications.

AngularJS **controllers** control AngularJS applications.

The **ng-app** directive defines the application, the **ng-controller** directive defines the controller.

AngularJS Example

```
<!DOCTYPE html>

<html>

<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>

<body>

<p>Try to change the names.</p>

<div ng-app="myApp" ng-controller="myCtrl">

First Name: <input type="text" ng-model="firstName"><br>

Last Name: <input type="text" ng-model="lastName"><br>

<br>

Full Name: {{firstName + " " + lastName}} </div>

<script>

var app = angular.module('myApp', []);

app.controller('myCtrl', function($scope) {

   $scope.firstName= "John";    $scope.lastName= "Doe"; }); </script> </body> </html>
```

AngularJS modules define applications:

AngularJS Module

```
var app = angular.module('myApp', []);
```

AngularJS controllers control applications:

AngularJS Controller

```
app.controller('myCtrl', function($scope) {
    $scope.firstName= "John";
    $scope.lastName= "Doe";
});
```

AngularJS Expressions

AngularJS Expressions

AngularJS expressions can be written inside double braces: `{{ expression }}`.

AngularJS expressions can also be written inside a directive: `ng-bind="expression"`.

AngularJS will resolve the expression, and return the result exactly where the expression is written.

AngularJS expressions are much like **JavaScript expressions:** They can contain literals, operators, and variables.

Example {{ 5 + 5 }} or {{ firstName + " " + lastName }}

Example

```
<!DOCTYPE html>
<html>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min
.js"></script>
<body>
```

```
<div ng-app="">
  <p>My first expression: {{ 5 + 5 }}</p>
</div>

</body>
</html>
```

Try it Yourself »

If you remove the `ng-app` directive, HTML will display the expression as it is, without solving it:

Example

```
<!DOCTYPE html>
<html>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min
.js"></script>
<body>

<div>
  <p>My first expression: {{ 5 + 5 }}</p>
</div>

</body>
</html>
```

Try it Yourself »

You can write expressions wherever you like, AngularJS will simply resolve the expression and return the result.

Example: Let AngularJS change the value of CSS properties.

Change the color of the input box below, by changing its value:

Example

```
<!DOCTYPE html>

<html>

<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>
```

<body>

<p>Change the value of the input field:</p>

<div ng-app="" ng-init="myCol='lightblue'">

<input style="background-color:{{myCol}}" ng-model="myCol" value="{{myCol}}">

</div>

<p>AngularJS resolves the expression and returns the result.</p>

<p>The background color of the input box will be whatever you write in the input field.</p>

</body> </html>

AngularJS Numbers

AngularJS numbers are like JavaScript numbers:

Example

<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>

<body> <div ng-app="" ng-init="quantity=1;cost=5">

<p>Total in dollar: {{ quantity * cost }}</p>

</div> </body> </html>

Same example using `ng-bind`:

Example

<body> <div ng-app="" ng-init="quantity=1;cost=5">

<p>Total in dollar: <span ng-bind="quantity * cost"></span></p>

</div> </body> </html>

AngularJS Strings

AngularJS strings are like JavaScript strings:

Example

```
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>

<body> <div ng-app="" ng-init="firstName='John';lastName='Doe'">

<p>The full name is: {{ firstName + " " + lastName }}</p> </div>
```

Same example using `ng-bind`:

Example

```
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>

<body> <div ng-app="" ng-init="firstName='John';lastName='Doe'">

<p>The full name is: <span ng-bind="firstName + ' ' + lastName"></span></p>

</div>
```

AngularJS Objects

AngularJS objects are like JavaScript objects:

Example

```
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>

<body> <div ng-app="" ng-init="person={firstName:'John',lastName:'Doe'}">

<p>The name is {{ person.lastName }}</p> </div>
```

Same example using `ng-bind`:

Example

```
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>

<body>

<div ng-app="" ng-init="person={firstName:'John',lastName:'Doe'}">

<p>The name is <span ng-bind="person.lastName"></span></p>

</div>
```

AngularJS Arrays

AngularJS arrays are like JavaScript arrays:

Example

```
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>

<body>

<div ng-app="" ng-init="points=[1,15,19,2,40]">

<p>The third result is {{ points[2] }}</p> </div>
```

Same example using `ng-bind`:

Example

```
<div ng-app="" ng-init="points=[1,15,19,2,40]">

<p>The third result is <span ng-bind="points[2]"></span></p>

</div>
```

AngularJS Expressions vs. JavaScript Expressions

Like JavaScript expressions, AngularJS expressions can contain literals, operators, and variables.

Unlike JavaScript expressions, AngularJS expressions can be written inside HTML.

AngularJS expressions do not support conditionals, loops, and exceptions, while JavaScript expressions do.

AngularJS expressions support filters, while JavaScript expressions do not.

AngularJS Modules

An AngularJS module defines an application.

The module is a container for the different parts of an application.

The module is a container for the application controllers.

Controllers always belong to a module.

Creating a Module

A module is created by using the AngularJS function `angular.module`

```
<div ng-app="myApp">...</div>

<script>

var app = angular.module("myApp", []);

</script>
```

The "myApp" parameter refers to an HTML element in which the application will run.

Now you can add controllers, directives, filters, and more, to your AngularJS application.

Adding a Controller

Add a controller to your application, and refer to the controller with the `ng-controller` directive:

Example

```
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>
```

```
<body>

<div ng-app="myApp" ng-controller="myCtrl">

{{ firstName + " " + lastName }}

</div>

<script>

var app = angular.module("myApp", []);

app.controller("myCtrl", function($scope) {

   $scope.firstName = "John";

   $scope.lastName = "Doe";

});

</script>
```

Adding a Directive

AngularJS has a set of built-in directives which you can use to add functionality to your application.

In addition you can use the module to add your own directives to your applications:

Example

```
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>

<body>

<div ng-app="myApp" w3-test-directive></div>

<script>

var app = angular.module("myApp", []);

app.directive("w3TestDirective", function() {

   return {

      template : "I was made in a directive constructor!"    }; }); </script>
```

Modules and Controllers in Files

It is common in AngularJS applications to put the module and the controllers in JavaScript files.

In this example, "myApp.js" contains an application module definition, while "myCtrl.js" contains the controller:

Example

```
<!DOCTYPE html>
<html>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min
.js"></script>
<body>

<div ng-app="myApp" ng-controller="myCtrl">
{{ firstName + " " + lastName }}
</div>

<script src="myApp.js"></script>
<script src="myCtrl.js"></script>

</body>
</html>
```

myApp.js

```
var app = angular.module("myApp", []);
```

The [] parameter in the module definition can be used to define dependent modules.

Without the [] parameter, you are not *creating* a new module, but *retrieving* an existing one.

myCtrl.js

```
app.controller("myCtrl", function($scope) {
    $scope.firstName = "John";
    $scope.lastName= "Doe";
});
```

Functions can Pollute the Global Namespace

Global functions should be avoided in JavaScript. They can easily be overwritten or destroyed by other scripts.

AngularJS modules reduces this problem, by keeping all functions local to the module.

When to Load the Library

While it is common in HTML applications to place scripts at the end of the `<body>` element, it is recommended that you load the AngularJS library either in the `<head>` or at the start of the `<body>`.

This is because calls to `angular.module` can only be compiled after the library has been loaded.

Example

```
<!DOCTYPE html>
<html>
<body>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min
.js"></script>

<div ng-app="myApp" ng-controller="myCtrl">
{{ firstName + " " + lastName }}
</div>

<script>
var app = angular.module("myApp", []);
app.controller("myCtrl", function($scope) {
    $scope.firstName = "John";
    $scope.lastName = "Doe";
});
```

```
</script>

</body>
</html>
```

AngularJS Directives

AngularJS lets you extend HTML with new attributes called **Directives**.

AngularJS has a set of built-in directives which offers functionality to your applications.

AngularJS also lets you define your own directives.

AngularJS Directives

AngularJS directives are extended HTML attributes with the prefix `ng-`.

The `ng-app` directive initializes an AngularJS application.

The `ng-init` directive initializes application data.

The `ng-model` directive binds the value of HTML controls (input, select, textarea) to application data.

Example

```
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>

<body>



<div ng-app="" ng-init="firstName='John'">



<p>Input something in the input box:</p>

<p>Name: <input type="text" ng-model="firstName"></p>

<p>You wrote: {{ firstName }}</p></div>
```

Data Binding

The `{{ firstName }}` expression, in the example above, is an AngularJS data binding expression.

Data binding in AngularJS binds AngularJS expressions with AngularJS data.

`{{ firstName }}` is bound with `ng-model="firstName"`.

In the next example two text fields are bound together with two ng-model directives:

Example

```
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>

<body>

<div data-ng-app="" data-ng-init="quantity=1;price=5">

<h2>Cost Calculator</h2>

Quantity: <input type="number" ng-model="quantity">

Price: <input type="number" ng-model="price">

<p><b>Total in dollar:</b> {{quantity * price}}</p> </div>
```

Using `ng-init` is not very common. You will learn how to initialize data in the chapter about controllers.

Repeating HTML Elements

The `ng-repeat` directive repeats an HTML element:

Example

```
<div ng-app="" ng-init="names=['Jani','Hege','Kai']">
  <ul>
    <li ng-repeat="x in names">
      {{ x }}    </li> </ul>
</div>
```

The `ng-repeat` directive actually **clones HTML elements** once for each item in a collection.

The `ng-repeat` directive used on an array of objects:

Example

```
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>

<body>

<div ng-app="" ng-init="names=[

{name:'Jani',country:'Norway'},

{name:'Hege',country:'Sweden'},

{name:'Kai',country:'Denmark'}]">

<p>Looping with objects:</p>

<ul>

  <li ng-repeat="x in names">

  {{ x.name + ', ' + x.country }}</li>

</ul> </div>
```

AngularJS is perfect for database CRUD (Create Read Update Delete) applications.
Just imagine if these objects were records from a database.

The ng-app Directive

The `ng-app` directive defines the **root element** of an AngularJS application.

The `ng-app` directive will **auto-bootstrap** (automatically initialize) the application when a web page is loaded.

The ng-init Directive

The `ng-init` directive defines **initial values** for an AngularJS application.

Normally, you will not use ng-init. You will use a controller or module instead.

You will learn more about controllers and modules later.

The ng-model Directive

The `ng-model` directive binds the value of HTML controls (input, select, textarea) to application data.

The `ng-model` directive can also:

- Provide type validation for application data (number, email, required).
- Provide status for application data (invalid, dirty, touched, error).
- Provide CSS classes for HTML elements.
- Bind HTML elements to HTML forms.

Read more about the `ng-model` directive in the next chapter.

Create New Directives

In addition to all the built-in AngularJS directives, you can create your own directives.

New directives are created by using the `.directive` function.

To invoke the new directive, make an HTML element with the same tag name as the new directive.

When naming a directive, you must use a camel case name, `w3TestDirective`, but when invoking it, you must use `-`separated name, `w3-test-directive`:

Example

```
<!DOCTYPE html>

<html>

<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>

<body ng-app="myApp">

<w3-test-directive></w3-test-directive>

<script>

var app = angular.module("myApp", []);

app.directive("w3TestDirective", function() {

    return {

        template : "<h1>Made by a directive!</h1>"

    };

});

</script></body> </html>
```

You can invoke a directive by using:

- Element name
- Attribute
- Class
- Comment

The examples below will all produce the same result:

Element name

```
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>

<body ng-app="myApp">
```

```
<w3-test-directive></w3-test-directive>

<script>

var app = angular.module("myApp", []);

app.directive("w3TestDirective", function() {

    return {

        template : "<h1>Made by a directive!</h1>"

    };

});

</script>
```

Attribute

```
<div w3-test-directive></div>
```

Class

```
<div class="w3-test-directive"></div>
```

Comment

```
<!-- directive: w3-test-directive -->
```

Restrictions

You can restrict your directives to only be invoked by some of the methods.

Example

By adding a `restrict` property with the value `"A"`, the directive can only be invoked by attributes:

```
<!DOCTYPE html>

<html> <script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>
```

```
<body ng-app="myApp">

<w3-test-directive></w3-test-directive>

<div w3-test-directive></div>

<script>

var app = angular.module("myApp", []);

app.directive("w3TestDirective", function() {

   return {

      restrict : "A",

      template : "<h1>Made by a directive!</h1>"

   };

});

</script>

<p><strong>Note:</strong> By setting the <strong>restrict</strong> property to "A", only the HTML
element with the "w3-test-directive" attribute has invoked the directive.</p>

</body></html>
```

The legal restrict values are:

- `E` for Element name
- `A` for Attribute
- `C` for Class
- `M` for Comment

By default the value is `EA`, meaning that both Element names and attribute names can invoke the directive

AngularJS ng-model Directive

The ng-model directive binds the value of HTML controls (input, select, textarea) to application data.

The ng-model Directive

With the `ng-model` directive you can bind the value of an input field to a variable created in AngularJS.

Example

The ng-model directive binds the value of HTML controls (input, select, textarea) to application data.

```
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>

<body> <div ng-app="myApp" ng-controller="myCtrl">

Name: <input ng-model="name">

</div><script>

var app = angular.module('myApp', []);

app.controller('myCtrl', function($scope) {

    $scope.name = "John Doe";

});

</script>

<p>Use the ng-model directive to bind the value of the input field to a property made in the controller.</p>
```

Two-Way Binding

The binding goes both ways. If the user changes the value inside the input field, the AngularJS property will also change its value:

Example

```
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>

<body>

<div ng-app="myApp" ng-controller="myCtrl">
```

```
Name: <input ng-model="name">

<h1>You entered: {{name}}</h1>

</div>

<script>

var app = angular.module('myApp', []);

app.controller('myCtrl', function($scope) {

   $scope.name = "John Doe";

});

</script>

<p>Change the name inside the input field, and you will see the name in the header changes
accordingly.</p>

</body>
```

Validate User Input

The `ng-model` directive can provide type validation for application data (number, e-mail, required):

Example

```
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>

<body>

<form ng-app="" name="myForm">

   Email:

   <input type="email" name="myAddress" ng-model="text">

   <span ng-show="myForm.myAddress.$error.email">Not a valid e-mail address</span>

</form>

<p>Enter your e-mail address in the input field. AngularJS will display an errormessage if the address is
not an e-mail.</p>
```

In the example above, the span will be displayed only if the expression in the `ng-show` attribute returns `true`. If the property in the `ng-model` attribute does not exist, AngularJS will create one for you.

Application Status

The `ng-model` directive can provide status for application data (invalid, dirty, touched, error):

Example

```
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>

<body>

<form ng-app="" name="myForm" ng-init="myText = 'post@myweb.com'">

Email:

<input type="email" name="myAddress" ng-model="myText" required>

<p>Edit the e-mail address, and try to change the status.</p>

<h1>Status</h1>

<p>Valid: {{myForm.myAddress.$valid}} (if true, the value meets all criteria).</p>

<p>Dirty: {{myForm.myAddress.$dirty}} (if true, the value has been changed).</p>

<p>Touched: {{myForm.myAddress.$touched}} (if true, the field has been in focus).</p>

</form>
```

CSS Classes

The `ng-model` directive provides CSS classes for HTML elements, depending on their status:

Example

```
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>

<style>

input.ng-invalid {

   background-color: lightblue;

}

</style>

<body> <form ng-app="" name="myForm">   Enter your name:
  <input name="myName" ng-model="myText" required>

</form>

<p>Edit the text field and it will get/lose classes according to the status.</p>

<p><b>Note:</b> A text field with the "required" attribute is not valid when it is empty.</p>

</body>
```

The `ng-model` directive adds/removes the following classes, according to the status of the form field:

- ng-empty
- ng-not-empty
- ng-touched
- ng-untouched
- ng-valid
- ng-invalid
- ng-dirty
- ng-pending
- ng-pristine

AngularJS Data Binding

Data binding in AngularJS is the synchronization between the model and the view.

Data Model

AngularJS applications usually have a data model. The data model is a collection of data available for the application.

Example

```
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope) {
    $scope.firstname = "John";
    $scope.lastname = "Doe";
});
```

HTML View

The HTML container where the AngularJS application is displayed, is called the view.

The view has access to the model, and there are several ways of displaying model data in the view.

You can use the `ng-bind` directive, which will bind the innerHTML of the element to the specified model property:

Example

The `ng-model` Directive

Use the `ng-model` directive to bind data from the model to the view on HTML controls (input, select, textarea)

Example

```
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>

<body><div ng-app="myApp" ng-controller="myCtrl">

    <input ng-model="firstname">

</div>

<script>

var app = angular.module('myApp', []);

app.controller('myCtrl', function($scope) {

    $scope.firstname = "John";

    $scope.lastname = "Doe";

});

</script>

<p>Use the ng-model directive on HTML controls (input, select, textarea) to bind data between the view
and the data model.</p> </body>
```

Two-way Binding

Data binding in AngularJS is the synchronization between the model and the view.

When data in the *model* changes, the *view* reflects the change, and when data in the *view* changes, the *model* is updated as well. This happens immediately and automatically, which makes sure that the model and the view is updated at all times.

Example

```
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>

<body> <div ng-app="myApp" ng-controller="myCtrl">
```

```
    Name: <input ng-model="firstname">

    <h1>{{firstname}}</h1>

</div>

<script>

var app = angular.module('myApp', []);

app.controller('myCtrl', function($scope) {

    $scope.firstname = "John";

    $scope.lastname = "Doe";

});

</script>

<p>Change the name inside the input field, and the model data will change automatically, and therefore
also the header will change its value.</p>

</body>
```

AngularJS Controller

Applications in AngularJS are controlled by controllers. Read about controllers in the AngularJS Controllers chapter.

Because of the immediate synchronization of the model and the view, the controller can be completely separated from the view, and simply concentrate on the model data. Thanks to the data binding in AngularJS, the view will reflect any changes made in the controller.

Example

```
<html>

<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>

<body>



<div ng-app="myApp" ng-controller="myCtrl">
```

```
    <h1 ng-click="changeName()">{{firstname}}</h1>

</div><script>

var app = angular.module('myApp', []);

app.controller('myCtrl', function($scope) {

    $scope.firstname = "John";

    $scope.changeName = function() {

        $scope.firstname = "Nelly";

    }

});

</script> <p>Click on the header to run the "changeName" function.</p>

<p>This example demonstrates how to use the controller to change model data.</p> </body> </html>
```

AngularJS Controllers

AngularJS controllers **control the data** of AngularJS applications.

AngularJS controllers are regular **JavaScript Objects**.

AngularJS Controllers

AngularJS applications are controlled by controllers.

The **ng-controller** directive defines the application controller.

A controller is a **JavaScript Object**, created by a standard JavaScript **object constructor**.

AngularJS Example

```
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>

<body>

<div ng-app="myApp" ng-controller="myCtrl">
```

```
First Name: <input type="text" ng-model="firstName"><br>

Last Name: <input type="text" ng-model="lastName"><br>

<br>

Full Name: {{firstName + " " + lastName}}

</div>

<script>

var app = angular.module('myApp', []);

app.controller('myCtrl', function($scope) {

    $scope.firstName = "John";

    $scope.lastName = "Doe";

});

</script>
```

Application explained:

The AngularJS application is defined by **ng-app="myApp"**. The application runs inside the <div>.

The **ng-controller="myCtrl"** attribute is an AngularJS directive. It defines a controller.

The **myCtrl** function is a JavaScript function.

AngularJS will invoke the controller with a **$scope** object.

In AngularJS, $scope is the application object (the owner of application variables and functions).

The controller creates two properties (variables) in the scope (**firstName** and **lastName**).

The **ng-model** directives bind the input fields to the controller properties (firstName and lastName).

Controller Methods

The example above demonstrated a controller object with two properties: lastName and firstName.

A controller can also have methods (variables as functions):

AngularJS Example

```
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>

<body>

<div ng-app="myApp" ng-controller="personCtrl">

First Name: <input type="text" ng-model="firstName"><br>

Last Name: <input type="text" ng-model="lastName"><br>

<br>

Full Name: {{fullName()}}

</div>

<script>

var app = angular.module('myApp', []);

app.controller('personCtrl', function($scope) {

    $scope.firstName = "John";

    $scope.lastName = "Doe";

    $scope.fullName = function() {

        return $scope.firstName + " " + $scope.lastName;

    };

});

</script>
```

Controllers In External Files

In larger applications, it is common to store controllers in external files.

Just copy the code between the <script> tags into an external file named personController.js:

AngularJS Example

<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>

<body>

<div ng-app="myApp" ng-controller="personCtrl">

First Name: <input type="text" ng-model="firstName"><br>

Last Name: <input type="text" ng-model="lastName"><br>

<br>

Full Name: {{fullName()}}

</div> <script src="personController.js"></script> </body>

Another Example

For the next example we will create a new controller file:

```
angular.module('myApp', []).controller('namesCtrl', function($scope) {
    $scope.names = [
        {name:'Jani',country:'Norway'},
        {name:'Hege',country:'Sweden'},
        {name:'Kai',country:'Denmark'}
    ];
});
```

AngularJS Example

<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>

<body>

<div ng-app="myApp" ng-controller="namesCtrl">

<ul>

  <li ng-repeat="x in names">

    {{ x.name + ', ' + x.country }}

  </li>

</ul> </div> <script src="namesController.js"></script> </body>

AngularJS Scope

The scope is the binding part between the HTML (view) and the JavaScript (controller).

The scope is an object with the available properties and methods.

The scope is available for both the view and the controller.

How to Use the Scope?

When you make a controller in AngularJS, you pass the `$scope` object as an argument:

Example

Properties made in the controller, can be referred to in the view:

```
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>

<body>

<div ng-app="myApp" ng-controller="myCtrl">

<h1>{{carname}}</h1>

</div>

<script>

var app = angular.module('myApp', []);

app.controller('myCtrl', function($scope) {

    $scope.carname = "Volvo";

});

</script>

<p>The property "carname" was made in the controller, and can be referred to in the view by using the {{
}} brackets.</p> </body>
```

When adding properties to the `$scope` object in the controller, the view (HTML) gets access to these properties.

In the view, you do not use the prefix `$scope`, you just refer to a propertyname, like `{{carname}}`.

Understanding the Scope

If we consider an AngularJS application to consist of:

- View, which is the HTML.
- Model, which is the data available for the current view.
- Controller, which is the JavaScript function that makes/changes/removes/controls the data.

Then the scope is the Model.

The scope is a JavaScript object with properties and methods, which are available for both the view and the controller.

Example

If you make changes in the view, the model and the controller will be updated:

```
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>

<body>

<div ng-app="myApp" ng-controller="myCtrl">

<input ng-model="name">

<h1>My name is {{name}}</h1> </div>

<script>

var app = angular.module('myApp', []);

app.controller('myCtrl', function($scope) {

    $scope.name = "John Doe";

});

</script>

<p>When you change the name in the input field, the changes will affect the model, and it will also affect
the name property in the controller.</p></body>
```

Know Your Scope

It is important to know which scope you are dealing with, at any time.

In the two examples above there is only one scope, so knowing your scope is not an issue, but for larger applications there can be sections in the HTML DOM which can only access certain scopes.

Example

When dealing with the `ng-repeat` directive, each repetition has access to the current repetition object:

```
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>

<body>

<div ng-app="myApp" ng-controller="myCtrl">

<ul>

   <li ng-repeat="x in names">{{x}}</li>

</ul> </div>

<script>

var app = angular.module('myApp', []);

app.controller('myCtrl', function($scope) {

   $scope.names = ["Emil", "Tobias", "Linus"];

});

</script>

<p>The variable "x" has a different value for each repetition, proving that each repetition has its own
scope.</p> </body>
```

Each `<li>` element has access to the current repetition object, in this case a string, which is referred to by using `x`.

Root Scope

All applications have a `$rootScope` which is the scope created on the HTML element that contains the `ng-app` directive.

The rootScope is available in the entire application.

If a variable has the same name in both the current scope and in the rootScope, the application use the one in the current scope.

Example

A variable named "color" exists in both the controller's scope and in the rootScope:

```
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>

<body ng-app="myApp">

<p>The rootScope's favorite color:</p>

<h1>{{color}}</h1>

<div ng-controller="myCtrl">

<p>The scope of the controller's favorite color:</p>

<h1>{{color}}</h1></div>

<p>The rootScope's favorite color is still:</p>

<h1>{{color}}</h1><script>

var app = angular.module('myApp', []);

app.run(function($rootScope) {

    $rootScope.color = 'blue';

});

app.controller('myCtrl', function($scope) {

    $scope.color = "red";

});</script> <p>Notice that controller's color variable does not overwrite the rootScope's color value.</p>
</body>
```

AngularJS Filters

Filters can be added in AngularJS to format data.

AngularJS Filters

AngularJS provides filters to transform data:

- `currency` Format a number to a currency format.
- `date` Format a date to a specified format.
- `filter` Select a subset of items from an array.
- `json` Format an object to a JSON string.
- `limitTo` Limits an array/string, into a specified number of elements/characters.
- `lowercase` Format a string to lower case.
- `number` Format a number to a string.
- `orderBy` Orders an array by an expression.
- `uppercase` Format a string to upper case.

Adding Filters to Expressions

Filters can be added to expressions by using the pipe character `|`, followed by a filter.

The `uppercase` filter format strings to upper case:

Example

```
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>

<body>

<div ng-app="myApp" ng-controller="personCtrl">

<p>The name is {{ lastName | uppercase }}</p>

</div>

<script>

angular.module('myApp', []).controller('personCtrl', function($scope) {

   $scope.firstName = "John",    $scope.lastName = "Doe" }); </script> </body>
```

he `lowercase` filter format strings to lower case:

Example

```
<div ng-app="myApp" ng-controller="personCtrl">

<p>The name is {{ lastName | lowercase }}</p>

</div>
```

Adding Filters to Directives

Filters are added to directives, like `ng-repeat`, by using the pipe character `|`, followed by a filter:

Example

The `orderBy` filter sorts an array:

```
<div ng-app="myApp" ng-controller="namesCtrl">

<ul>
  <li ng-repeat="x in names | orderBy:'country'">
    {{ x.name + ', ' + x.country }}
  </li>
</ul>

</div>
```

The currency Filter

The `currency` filter formats a number as currency:

Example

```
<div ng-app="myApp" ng-controller="costCtrl">

<h1>Price: {{ price | currency }}</h1>

</div>
```

The filter Filter

The `filter` filter selects a subset of an array.

The `filter` filter can only be used on arrays, and it returns an array containing only the matching items.

Example

Return the names that contains the letter "i":

```
<div ng-app="myApp" ng-controller="namesCtrl">

<ul>
  <li ng-repeat="x in names | filter : 'i'">
    {{ x }}
  </li>
</ul>

</div>
```

Filter an Array Based on User Input

By setting the `ng-model` directive on an input field, we can use the value of the input field as an expression in a filter.

Type a letter in the input field, and the list will shrink/grow depending on the match:

- Jani
- Carl
- Margareth
- Hege
- Joe
- Gustav
- Birgit
- Mary
- Kai

Example

```
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>

<body> <div ng-app="myApp" ng-controller="namesCtrl">

<p>Type a letter in the input field:</p>

<p><input type="text" ng-model="test"></p>

<ul>

  <li ng-repeat="x in names | filter:test">  {{ x }}  </li></ul></div>
```

<script>

angular.module('myApp', []).controller('namesCtrl', function($scope) {

    $scope.names = ['Jani', 'Carl', 'Margareth', 'Hege','Joe','Gustav','Birgit','Mary', 'Kai']});

</script>

<p>The list will only consists of names matching the filter.</p>

</body>

Sort an Array Based on User Input

Click the table headers to change the sort order::

Name	**Country**
Jani	Norway
Carl	Sweden
Margareth	England
Hege	Norway
Joe	Denmark
Gustav	Sweden
Birgit	Denmark
Mary	England

Kai	Norway

By adding the `ng-click` directive on the table headers, we can run a function that changes the sorting order of the array:

Example

```
<!DOCTYPE html>

<html>

<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>

<body>

<p>Click the table headers to change the sorting order:</p>

<div ng-app="myApp" ng-controller="namesCtrl">

<table border="1" width="100%">

<tr>

<th ng-click="orderByMe('name')">Name</th>

<th ng-click="orderByMe('country')">Country</th>

</tr>

<tr ng-repeat="x in names | orderBy:myOrderBy">

<td>{{x.name}}</td>

<td>{{x.country}}</td>

</tr>

</table></div>

<script>

angular.module('myApp', []).controller('namesCtrl', function($scope) {

    $scope.names =
[{name:'Jani',country:'Norway'},{name:'Carl',country:'Sweden'},{name:'Margareth',country:'England'},
{name:'Hege',country:'Norway'},{name:'Joe',country:'Denmark'},{name:'Gustav',country:'Sweden'},

{name:'Birgit',country:'Denmark'},{name:'Mary',country:'England'},{name:'Kai',country:'Norway'}];

    $scope.orderByMe = function(x) {

        $scope.myOrderBy = x;

    }

});

</script></body></html>
```

Custom Filters

You can make your own filters by registering a new filter factory function with your module:

Example

Make a custom filter called "myFormat":

```
<html>

<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>

<body>

<ul ng-app="myApp" ng-controller="namesCtrl">

<li ng-repeat="x in names">

    {{x | myFormat}}</li></ul>

<script>

var app = angular.module('myApp', []);

app.filter('myFormat', function() {

    return function(x) {

        var i, c, txt = "";

        for (i = 0; i < x.length; i++) {
            c = x[i];

            if (i % 2 == 0) {

                c = c.toUpperCase();

            }

            txt += c;

        }

        return txt;

    };

});

app.controller('namesCtrl', function($scope) {

    $scope.names = ['Jani','Carl',  'Margareth','Hege','Joe','Gustav','Birgit','Mary',  'Kai' ];});

</script>

<p>Make your own filters.</p>

<p>This filter, called "myFormat", will uppercase every other character.</p>

</body>
```

AngularJS Services

In AngularJS you can make your own service, or use one of the many built-in services.

What is a Service?

In AngularJS, a service is a function, or object, that is available for, and limited to, your AngularJS application.

AngularJS has about 30 built-in services. One of them is the `$location` service.

The `$location` service has methods which return information about the location of the current web page:

Example

Use the `$location` service in a controller:

```
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>

<body>

<div ng-app="myApp" ng-controller="myCtrl">

<p>The url of this page is:</p>

<h3>{{myUrl}}</h3>

</div>

<p>This example uses the built-in $location service to get the absolute url of the page.</p>

<script>

var app = angular.module('myApp', []);

app.controller('myCtrl', function($scope, $location) {

   $scope.myUrl = $location.absUrl();

});

</script>
```

Note that the `$location` service is passed in to the controller as an argument. In order to use the service in the controller, it must be defined as a dependency.

Why use Services?

For many services, like the `$location` service, it seems like you could use objects that are already in the DOM, like the`window.location` object, and you could, but it would have some limitations, at least for your AngularJS application.

AngularJS constantly supervises your application, and for it to handle changes and events properly, AngularJS prefers that you use the `$location` service instead of the `window.location` object.

The $http Service

The `$http` service is one of the most common used services in AngularJS applications. The service makes a request to the server, and lets your application handle the response.

Example

Use the `$http` service to request data from the server:

```
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>
<body>
<div ng-app="myApp" ng-controller="myCtrl">
<p>Today's welcome message is:</p>
<h1>{{myWelcome}}</h1>
</div>
<p>The $http service requests a page on the server, and the response is set as the value of the
"myWelcome" variable.</p>
<script>
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope, $http) {
  $http.get("welcome.htm").then(function (response) {
      $scope.myWelcome = response.data;
  });
});
</script>
```

The $timeout Service

The `$timeout` service is AngularJS' version of the `window.setTimeout` function.

Example

Display a new message after two seconds:

```
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>
<body>
<div ng-app="myApp" ng-controller="myCtrl">
```

```
<p>This header will change after two seconds:</p>

<h1>{{myHeader}}</h1>

</div>

<p>The $timeout service runs a function after a sepecified number of milliseconds.</p>

<script>

var app = angular.module('myApp', []);

app.controller('myCtrl', function($scope, $timeout) {

  $scope.myHeader = "Hello World!";

  $timeout(function () {

      $scope.myHeader = "How are you today?";

  }, 2000);

});

</script>
```

The $interval Service

The `$interval` service is AngularJS' version of the `window.setInterval` function.

Example

```
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>

<body>

<div ng-app="myApp" ng-controller="myCtrl">

<p>The time is:</p>

<h1>{{theTime}}</h1>

</div>

<p>The $interval service runs a function every specified millisecond.</p>

<script>

var app = angular.module('myApp', []);
```

app.controller('myCtrl', function($scope, $interval) {

$scope.theTime = new Date().toLocaleTimeString();

$interval(function () {

$scope.theTime = new Date().toLocaleTimeString();

}, 1000);

});

</script> </body>

Create Your Own Service

To create your own service, connect your service to the module:

Create a service named hexafy:

```
app.service('hexafy', function() {
    this.myFunc = function (x) {
        return x.toString(16);
    }
});
```

To use your custom made service, add it as a dependency when defining the filter:

Example

Use the custom made service named hexafy to convert a number into a hexadecimal number:

<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>

<body>

<div ng-app="myApp" ng-controller="myCtrl">

<p>The hexadecimal value of 255 is:</p>

<h1>{{hex}}</h1>

</div>

<p>A custom service whith a method that converts a given number into a hexadecimal number.</p>

<script>

```
var app = angular.module('myApp', []);

app.service('hexafy', function() {

    this.myFunc = function (x) {

        return x.toString(16);

    }

});

app.controller('myCtrl', function($scope, hexafy) {

  $scope.hex = hexafy.myFunc(255);

});

</script>
```

Use a Custom Service Inside a Filter

Once you have created a service, and connected it to your application, you can use the service in any controller, directive, filter, or even inside other services.

To use the service inside a filter, add it as a dependency when defining the filter:

The service `hexafy` used in the filter `myFormat`:

```
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>

<body> <div ng-app="myApp">

Convert the number 255, using a custom made service inside a custom made filter:

<h1>{{255 | myFormat}}</h1>

</div> <script>

var app = angular.module('myApp', []);

app.service('hexafy', function() {

    this.myFunc = function (x) {

        return x.toString(16);

    }
});

app.filter('myFormat',['hexafy', function(hexafy) {

    return function(x) {

        return hexafy.myFunc(x);

    };

}]);

</script>
```

AngularJS Tables

The ng-repeat directive is perfect for displaying tables.

Displaying Data in a Table

Displaying tables with angular is very simple:

AngularJS Example

```
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>

<body>

<div ng-app="myApp" ng-controller="customersCtrl">

<table>

  <tr ng-repeat="x in names">

    <td>{{ x.Name }}</td>

    <td>{{ x.Country }}</td>

  </tr>

</table> </div>

<script>

var app = angular.module('myApp', []);

app.controller('customersCtrl', function($scope, $http) {
    $http.get("http://www.w3schools.com/angular/customers.php")

    .then(function (response) {$scope.names = response.data.records;});

});

</script>
```

Displaying with CSS Style

To make it nice, add some CSS to the page:

CSS Style

```
<style>

table, th , td  {

  border: 1px solid grey;

  border-collapse: collapse;

  padding: 5px;

}

table tr:nth-child(odd) {

  background-color: #f1f1f1;

}

table tr:nth-child(even) {

  background-color: #ffffff;

}

</style>

<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>

<body>

<div ng-app="myApp" ng-controller="customersCtrl">

<table>

  <tr ng-repeat="x in names">
```

```
   <td>{{ x.Name }}</td>
   <td>{{ x.Country }}</td>
  </tr>
</table> </div>
<script>
var app = angular.module('myApp', []);
app.controller('customersCtrl', function($scope, $http) {
   $http.get("http://www.w3schools.com/angular/customers.php")
   .then(function (response) {$scope.names = response.data.records;});
});
</script>
```

Display with orderBy Filter

To sort the table, add an **orderBy** filter:

```
<style>
table, th , td  {
  border: 1px solid grey;
  border-collapse: collapse;
  padding: 5px;
}
table tr:nth-child(odd) {
  background-color: #f1f1f1;
}
table tr:nth-child(even) {
  background-color: #ffffff;
}
```

```
</style>

<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>

<body>

<div ng-app="myApp" ng-controller="customersCtrl">

<table>

  <tr ng-repeat="x in names | orderBy : 'Country'">

    <td>{{ x.Name }}</td>

    <td>{{ x.Country }}</td>

  </tr>

</table>

</div>

<script>

var app = angular.module('myApp', []);

app.controller('customersCtrl', function($scope, $http) {

    $http.get("http://www.w3schools.com/angular/customers.php")

    .then(function (response) {$scope.names = response.data.records;});

});

</script>
```

AngularJS Select Boxes

AngularJS lets you create dropdown lists based on items in an array, or an object.

Creating a Select Box Using ng-options

If you want to create a dropdown list, based on an object or an array in AngularJS, you should use the `ng-options`directive:

Example

```
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>
```

```
<body> <div ng-app="myApp" ng-controller="myCtrl">

<select ng-model="selectedName" ng-options="x for x in names">

</select> </div>

<script>

var app = angular.module('myApp', []);

app.controller('myCtrl', function($scope) {

    $scope.names = ["Emil", "Tobias", "Linus"];

});

</script>

<p>This example shows how to fill a dropdown list using the ng-options directive.</p>

</body>
```

ng-options vs ng-repeat

You can also use the `ng-repeat` directive to make the same dropdown list:

Example

```
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>

<body>

<div ng-app="myApp" ng-controller="myCtrl">

<select>

<option ng-repeat="x in names">{{x}}</option>

</select> </div>

<script>

var app = angular.module('myApp', []);

app.controller('myCtrl', function($scope) {

   $scope.names = ["Emil", "Tobias", "Linus"];
```

}); </script> <p>This example shows how to fill a dropdown list using the ng-repeat directive.</p>
</body>

Because the `ng-repeat` directive repeats a block of HTML code for each item in an array, it can be used to create options in a dropdown list, but the `ng-options` directive was made especially for filling a dropdown list with options, and has at least one important advantage:

Dropdowns made with `ng-options` allows the selected value to be an **object**, while dropdowns made from `ng-repeat`has to be a string.

What Do I Use?

Assume you have an array of objects:

```
$scope.cars = [
    {model : "Ford Mustang", color : "red"},
    {model : "Fiat 500", color : "white"},
    {model : "Volvo XC90", color : "black"}
];
```

The `ng-repeat` directive has its limitations, the selected value must be a string:

Example

```
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>

<body>

<div ng-app="myApp" ng-controller="myCtrl">

<p>Select a car:</p>

<select ng-model="selectedCar">

<option ng-repeat="x in cars" value="{{x.model}}">{{x.model}}</option>

</select>

<h1>You selected: {{selectedCar}}</h1>

</div>

<script>

var app = angular.module('myApp', []);

app.controller('myCtrl', function($scope) {
   $scope.cars = [
       {model : "Ford Mustang", color : "red"},
       {model : "Fiat 500", color : "white"},
       {model : "Volvo XC90", color : "black"}
   ];
});
</script>
<p>When you use the ng-repeat directive to create dropdown lists, the selected value must be a string.</p>
<p>In this example you will have to choose between the color or the model to be your selected value.</p>
</body>
```

When using the `ng-options` directive, the selected value can be an object:

Example

```
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>
<body>
<div ng-app="myApp" ng-controller="myCtrl">
<p>Select a car:</p>
<select ng-model="selectedCar" ng-options="x.model for x in cars">
</select>
<h1>You selected: {{selectedCar.model}}</h1>
<p>Its color is: {{selectedCar.color}}</p> </div>
<script>
var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope) {
   $scope.cars = [
```

{model : "Ford Mustang", color : "red"},

{model : "Fiat 500", color : "white"},

{model : "Volvo XC90", color : "black"}

];

});

</script>

<p>When you use the ng-options directive to create dropdown lists, the selected value can be an object.</p>

<p>In this example you can display both the model and the color of the selected element.</p> </body>

The Data Source as an Object

In the previous examples the data source was an array, but we can also use an object.

Assume you have an object with key-value pairs:

```
$scope.cars = {
    car01 : "Ford",
    car02 : "Fiat",
    car03 : "Volvo"
};
```

The expression in the `ng-options` attribute is a bit different for objects:

Example:

<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>

<body>

<div ng-app="myApp" ng-controller="myCtrl">

<p>Select a car:</p>

<select ng-model="selectedCar" ng-options="x for (x, y) in cars">

</select>

<h1>You selected: {{selectedCar}}</h1>

</div>

```
<p>This example demonstrates the use of an object as the data source when creating a dropdown list.</p>

<script>

var app = angular.module('myApp', []);

app.controller('myCtrl', function($scope) {

   $scope.cars = {

      car01 : "Ford",

      car02 : "Fiat",

      car03 : "Volvo"

   }

});

</script>
```

AngularJS Events

AngularJS Events

You can add AngularJS event listeners to your HTML elements by using one or more of these directives:

- `ng-blur`
- `ng-change`
- `ng-click`
- `ng-copy`
- `ng-cut`
- `ng-dblclick`
- `ng-focus`
- `ng-keydown`
- `ng-keypress`
- `ng-keyup`
- `ng-mousedown`
- `ng-mouseenter`
- `ng-mouseleave`
- `ng-mousemove`
- `ng-mouseover`
- `ng-mouseup`
- `ng-paste`

The event directives allows us to run AngularJS functions at certain user events.

An AngularJS event will not overwrite an HTML event, both events will be executed.

Mouse Events

Mouse events occur when the cursor moves over an element, in this order:

1. ng-mouseenter
2. ng-mouseover
3. ng-mousemove
4. ng-mouseleave

Or when a mouse button is clicked on an element, in this order:

1. ng-mousedown
2. ng-mouseup
3. ng-click

You can add mouse events on any HTML element.

Example

```
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>

<body>

<div ng-app="myApp" ng-controller="myCtrl">

<h1 ng-mousemove="count = count + 1">Mouse Over Me!</h1>

<h2>{{ count }}</h2> </div>

<script>

var app = angular.module('myApp', []);

app.controller('myCtrl', function($scope) {

   $scope.count = 0;

});

</script>
```

The ng-click Directive

The `ng-click` directive defines AngularJS code that will be executed when the element is being clicked.

Example

```
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>

<body>

<div ng-app="myApp" ng-controller="myCtrl">

<button ng-click="count = count + 1">Click Me!</button>

<p>{{ count }}</p>

</div>

<script>

var app = angular.module('myApp', []);

app.controller('myCtrl', function($scope) {

   $scope.count = 0;

});

</script>
```

Toggle, True/False

If you want to show a section of HTML code when a button is clicked, and hide when the button is clicked again, like a dropdown menu, make the button behave like a toggle switch:

Example

```
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>

<body>

<div ng-app="myApp" ng-controller="myCtrl">

<button ng-click="myFunc()">Click Me!</button>

<div ng-show="showMe">

   <h1>Menu:</h1>
```

```
  <div>Pizza</div>

  <div>Pasta</div>

  <div>Pesce</div>

</div>

</div>

<script>

var app = angular.module('myApp', []);

app.controller('myCtrl', function($scope) {

  $scope.showMe = false;

  $scope.myFunc = function() {

    $scope.showMe = !$scope.showMe;

  }

});

</script>

<p>Click the button to show/hide the menu.</p>

</body>
```

The `showMe` variable starts out as the Boolean value `false`.

The `myFunc` function sets the `showMe` variable to the opposite of what it is, by using the `!` (not) operator.

$event Object

You can pass the `$event` object as an argument when calling the function.

The `$event` object contains the browser's event object:

Example

```
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>

<body> <div ng-app="myApp" ng-controller="myCtrl">
```

```
<h1 ng-mousemove="myFunc($event)">Mouse Over Me!</h1>

<p>Coordinates: {{x + ', ' + y}}</p>

</div>

<script>

var app = angular.module('myApp', []);

app.controller('myCtrl', function($scope) {

   $scope.myFunc = function(myE) {

      $scope.x = myE.clientX;

      $scope.y = myE.clientY;

   }

});

</script>

<p>Mouse over the heading to display the value of clientX and clientY from the event object.</p>

</body>
```

AngularJS Forms

Input Controls

Input controls are the HTML input elements:

- input elements
- select elements
- button elements
- textarea elements

Data-Binding

Input controls provides data-binding by using the `ng-model` directive.

```
<input type="text" ng-model="firstname">
```

The application does now have a property named `firstname`.

The `ng-model` directive binds the input controller to the rest of your application.

The property `firstname`, can be referred to in a controller:

Example

```
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>

<body>

<div ng-app="myApp" ng-controller="formCtrl">

  <form>

    First Name: <input type="text" ng-model="firstname">

  </form>

</div>

<script>

var app = angular.module('myApp', []);

app.controller('formCtrl', function($scope) {

    $scope.firstname = "John";

});

</script>
```

Checkbox

A checkbox has the value `true` or `false`. Apply the `ng-model` directive to a checkbox, and use its value in your application.

Example

```
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>

<body>

<div ng-app="">

  <form>
```

```
    Check to show a header:

    <input type="checkbox" ng-model="myVar">

  </form>

  <h1 ng-show="myVar">My Header</h1>

</div>

<p>The header's ng-show attribute is set to true when the checkbox is checked.</p>

</body>
```

Radiobuttons

Bind radio buttons to your application with the `ng-model` directive.

Radio buttons with the same `ng-model` can have different values, but only the selected one will be used.

Example

```
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>

<body ng-app="">

<form>

  Pick a topic:

  <input type="radio" ng-model="myVar" value="dogs">Dogs

  <input type="radio" ng-model="myVar" value="tuts">Tutorials

  <input type="radio" ng-model="myVar" value="cars">Cars

</form>

<div ng-switch="myVar">

  <div ng-switch-when="dogs">

     <h1>Dogs</h1>

     <p>Welcome to a world of dogs.</p>

  </div>

  <div ng-switch-when="tuts">
```

```
    <h1>Tutorials</h1>

    <p>Learn from examples.</p>

  </div>

  <div ng-switch-when="cars">

    <h1>Cars</h1>

    <p>Read about cars.</p>

  </div>

</div>

<p>The ng-switch directive hides and shows HTML sections depending on the value of the radio buttons.</p>

</body>
```

Selectbox

Bind select boxes to your application with the `ng-model` directive.

The property defined in the `ng-model` attribute will have the value of the selected option in the selectbox.

Example

```
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>

<body ng-app="">

<form>

  Select a topic:

  <select ng-model="myVar">

    <option value="">

    <option value="dogs">Dogs

    <option value="tuts">Tutorials

    <option value="cars">Cars

  </select>
```

```
</form>

<div ng-switch="myVar">

  <div ng-switch-when="dogs">

     <h1>Dogs</h1>

     <p>Welcome to a world of dogs.</p>

  </div>

  <div ng-switch-when="tuts">

     <h1>Tutorials</h1>

     <p>Learn from examples.</p>

  </div>

  <div ng-switch-when="cars">

     <h1>Cars</h1>

     <p>Read about cars.</p>

  </div>

</div>

<p>The ng-switch directive hides and shows HTML sections depending on the value of the dropdown
list.</p>

</body>
```

An AngularJS Form Example

Application Code

```
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>

<body>

<div ng-app="myApp" ng-controller="formCtrl">

  <form novalidate>

    First Name:<br>
    <input type="text" ng-model="user.firstName"><br>
    Last Name:<br>
    <input type="text" ng-model="user.lastName">
    <br><br>
    <button ng-click="reset()">RESET</button>
  </form>
  <p>form = {{user}}</p>
  <p>master = {{master}}</p>
</div>
<script>
var app = angular.module('myApp', []);
app.controller('formCtrl', function($scope) {
    $scope.master = {firstName:"John", lastName:"Doe"};
    $scope.reset = function() {
        $scope.user = angular.copy($scope.master);
    };
    $scope.reset();
});
</script>
```

Example Explained

The **ng-app** directive defines the AngularJS application.

The **ng-controller** directive defines the application controller.

The **ng-model** directive binds two input elements to the **user** object in the model.

The **formCtrl** controller sets initial values to the **master** object, and defines the **reset()** method.

The **reset()** method sets the **user** object equal to the **master** object.

The **ng-click** directive invokes the **reset()** method, only if the button is clicked.

The novalidate attribute is not needed for this application, but normally you will use it in AngularJS forms, to override standard HTML5 validation.

AngularJS Form Validation

Form Validation

AngularJS offers client-side form validation.

AngularJS monitors the state of the form and input fields (input, textarea, select), and lets you notify the user about the current state.

AngularJS also holds information about whether they have been touched, or modified, or not.

You can use standard HTML5 attributes to validate input, or you can make your own validation functions.

Required

Use the HTML5 attribute `required` to specify that the input field must be filled out:

Example

```
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>

<body ng-app="">

<p>Try writing in the input field:</p>

<form name="myForm">

<input name="myInput" ng-model="myInput" required>

</form>

<p>The input's valid state is:</p>

<h1>{{myForm.myInput.$valid}}</h1>

</body>
```

E-mail

Use the HTML5 type `email` to specify that the value must be an e-mail:

Example

```
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>

<body ng-app="">

<p>Try writing an E-mail address in the input field:</p>

<form name="myForm">

<input type="email" name="myInput" ng-model="myInput">

</form>

<p>The input's valid state is:</p>

<h1>{{myForm.myInput.$valid}}</h1>

<p>Note that the state of the input field is "true" before you start writing in it, even if it does not contain an e-mail address.</p>

</body>
```

Form State and Input State

AngularJS is constantly updating the state of both the form and the input fields.

Input fields have the following states:

- `$untouched` The field has not been touched yet
- `$touched` The field has been touched
- `$pristine` The field has not been modified yet
- `$dirty` The field has been modified
- `$invalid` The field content is not valid
- `$valid` The field content is valid

They are all properties of the input field, and are either `true` or `false`.

Forms have the following states:

- `$pristine` No fields have been modified yet
- `$dirty` One or more have been modified

- `$invalid` The form content is not valid
- `$valid` The form content is valid
- `$submitted` The form is submitted

They are all properties of the form, and are either `true` or `false`.

You can use these states to show meaningful messages to the user. Example, if a field is required, and the user leaves it blank, you should give the user a warning:

Example

```
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>

<body ng-app=""> <p>Try leaving the first input field blank:</p>

<form name="myForm">

<p>Name:

<input name="myName" ng-model="myName" required>

<span ng-show="myForm.myName.$touched && myForm.myName.$invalid">The name is required.</span>

</p>

<p>Adress:

<input name="myAddress" ng-model="myAddress" required>

</p>

</form>

<p>We use the ng-show directive to only show the error message if the field has been touched AND is empty.</p>

</body>
```

CSS Classes

AngularJS adds CSS classes to forms and input fields depending on their states.

The following classes are added to, or removed from, input fields:

- `ng-untouched` The field has not been touched yet

- `ng-touched` The field has been touched
- `ng-pristine` The field has not been modified yet
- `ng-dirty` The field has been modified
- `ng-valid` The field content is valid
- `ng-invalid` The field content is not valid
- `ng-valid-*key*` One *key* for each validation. Example: `ng-valid-required`, useful when there are more than one thing that must be validated
- `ng-invalid-*key*` Example: `ng-invalid-required`

The following classes are added to, or removed from, forms:

- `ng-pristine` No fields has not been modified yet
- `ng-dirty` One or more fields has been modified
- `ng-valid` The form content is valid
- `ng-invalid` The form content is not valid
- `ng-valid-*key*` One *key* for each validation. Example: `ng-valid-required`, useful when there are more than one thing that must be validated
- `ng-invalid-*key*` Example: `ng-invalid-required`

The classes are removed if the value they represent is `false`.

Add styles for these classes to give your application a better and more intuitive user interface.

Example

```
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>

<style>

input.ng-invalid {

    background-color:pink;

}

input.ng-valid {

    background-color:lightgreen;

}

</style>

<body ng-app="">

<p>Try writing in the input field:</p>
```

```
<form name="myForm">

<input name="myName" ng-model="myName" required>

</form>

<p>The input field requires content, and will therefore become green when you write in it.</p>

</body>
```

Custom Validation

To create your own validation function is a bit more tricky. You have to add a new directive to your application, and deal with the validation inside a function with certain specified arguments.

Example

Create your own directive, containing a custom validation function, and refer to it by using `my-directive`.

The field will only be valid if the value contains the character "e":

```
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>

<body ng-app="myApp">

<p>Try writing in the input field:</p>

<form name="myForm">

<input name="myInput" ng-model="myInput" required my-directive>

</form>

<p>The input's valid state is:</p>

<h1>{{myForm.myInput.$valid}}</h1>

<script>

var app = angular.module('myApp', []);

app.directive('myDirective', function() {

   return {

      require: 'ngModel',
        link: function(scope, element, attr, mCtrl) {

            function myValidation(value) {

                if (value.indexOf("e") > -1) {

                    mCtrl.$setValidity('charE', true);

                } else {

                    mCtrl.$setValidity('charE', false);

                }

                return value;

            }

            mCtrl.$parsers.push(myValidation);

        }

    };

});

</script>

<p>The input field must contain the character "e" to be consider valid.</p>

</body>
```

Example Explained:

In HTML, the new directive will be referred to by using the attribute `my-directive`.

In the JavaScript we start by adding a new directive named `myDirective`.

Remember, when naming a directive, you must use a camel case name, `myDirective`, but when invoking it, you must use `-` separated name, `my-directive`.

Then, return an object where you specify that we require `ngModel`, which is the ngModelController.

Make a linking function which takes some arguments, where the fourth argument, `mCtrl`, is the `ngModelController`, Then specify a function, in this case named `myValidation`, which takes one argument, this argument is the value of the input element.

Test if the value contains the letter "e", and set the validity of the model controller to either `true` or `false`.

At last, `mCtrl.$parsers.push(myValidation);` will add the `myValidation` function to an array of other functions, which will be executed every time the input value changes.

Validation Example

```
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>

<body>

<h2>Validation Example</h2>

<form ng-app="myApp" ng-controller="validateCtrl"

name="myForm" novalidate>

<p>Username:<br>

<input type="text" name="user" ng-model="user" required>

<span style="color:red" ng-show="myForm.user.$dirty && myForm.user.$invalid">

<span ng-show="myForm.user.$error.required">Username is required.</span>

</span>

</p>

<p>Email:<br>

<input type="email" name="email" ng-model="email" required>

<span style="color:red" ng-show="myForm.email.$dirty && myForm.email.$invalid">

<span ng-show="myForm.email.$error.required">Email is required.</span>

<span ng-show="myForm.email.$error.email">Invalid email address.</span>

</span>

</p>
```

```
<p>

<input type="submit"

ng-disabled="myForm.user.$dirty && myForm.user.$invalid ||

myForm.email.$dirty && myForm.email.$invalid">

</p>

</form>

<script>

var app = angular.module('myApp', []);

app.controller('validateCtrl', function($scope) {

    $scope.user = 'John Doe';

    $scope.email = 'john.doe@gmail.com';

});

</script>
```

Example Explained

The AngularJS directive **ng-model** binds the input elements to the model.

The model object has two properties: **user** and **email**.

Because of **ng-show**, the spans with color:red are displayed only when user or email is **$dirty** and **$invalid**.

AngularJS API

AngularJS Global API

The AngularJS Global API is a set of global JavaScript functions for performing common tasks like:

- Comparing objects
- Iterating objects
- Converting data

The Global API functions are accessed using the angular object.

Below is a list of some common API functions:

API	Description
angular.lowercase()	Converts a string to lowercase
angular.uppercase()	Converts a string to uppercase
angular.isString()	Returns true if the reference is a string
angular.isNumber()	Returns true if the reference is a number

angular.lowercase()

Example

```
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>

<body>



<div ng-app="myApp" ng-controller="myCtrl">

<p>{{ x1 }}</p>

<p>{{ x2 }}</p>

</div>



<script>

var app = angular.module('myApp', []);

app.controller('myCtrl', function($scope) {

   $scope.x1 = "JOHN";

   $scope.x2 = angular.lowercase($scope.x1);
});

</script>
```

angular.uppercase()

Example

```
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>

<body>

<div ng-app="myApp" ng-controller="myCtrl">

<p>{{ x1 }}</p>

<p>{{ x2 }}</p>

</div>

<script>

var app = angular.module('myApp', []);

app.controller('myCtrl', function($scope) {

   $scope.x1 = "John";

   $scope.x2 = angular.uppercase($scope.x1);

});

</script>
```

angular.isString()

Example

```
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>

<body>

<div ng-app="myApp" ng-controller="myCtrl">

<p>{{ x1 }}</p>

<p>{{ x2 }}</p>

</div>
```

```
<script>

var app = angular.module('myApp', []);

app.controller('myCtrl', function($scope) {

   $scope.x1 = "JOHN";

   $scope.x2 = angular.isString($scope.x1);

});

</script>
```

angular.isNumber()

Example

```
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular.min.js"></script>


<body>


<div ng-app="myApp" ng-controller="myCtrl">

<p>{{ x1 }}</p>

<p>{{ x2 }}</p>

</div>

<script>

var app = angular.module('myApp', []);

app.controller('myCtrl', function($scope) {

   $scope.x1 = "JOHN";

   $scope.x2 = angular.isNumber($scope.x1);

});

</script>
```

s